BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

ポイント図解式

通信プロトコル事典

PROTOCOL

笠野英松◆監修
マルチメディア通信研究会◆編

アスキー出版局

ィア文書と呼んでもかまいません）を構築できるようになっています。

HyTimeを使うことで具体的に次のような機能をSGMLに付加することができます。

① 計測単位の提供（秒やcmなど）
② 座標軸の提供（XYZTなど）
③ 文書要素（オブジェクトと呼んでもかまいません）の名前づけ方法
④ オブジェクトのさまざまなアドレッシング方法の提供（例えば、オブジェクトの開始から10秒後などという指定）
⑤ オブジェクト間にハイパーリンクを張る方法
⑥ 特別なレイアウト指示（あるオブジェクトの投影や変形を提供する機能）

HyTimeを使ったアプリケーションはまだあまり多く出回っていませんが、HTMLの普及にともなって注目される技術であろうと思われます。HTMLが時変要素を本格的に扱うことになるならば、無駄な標準の乱立を避けるためにも、ぜひ、HyTimeを採用、あるいは参考にして欲しいものだと思います。

10.4 インターネットのマルチメディア情報通信プロトコル-HTTP/1.0-

HTTPの概要

(1) HTTPとWWW

HTTP（HyperText Transfer Protocol、ハイパーテキスト転送プロトコル）は、WWW（World Wide Web、ワールド・ワイド・ウェブ、図10-26）と呼ばれる世界的な情報網として、1990年に使用されはじめたインターネットのアプ

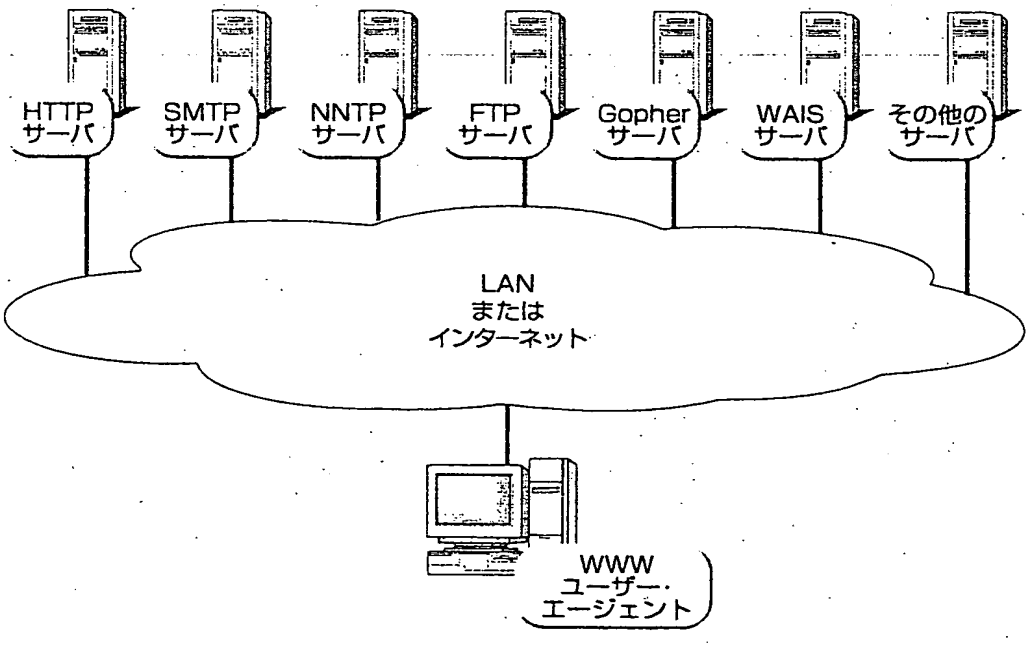


図10-26 WWWと各種プロトコルのサーバ

ポイント 図10-26

❶ WWWはHTTP以外のプロトコルも含む

HTTPは、分散環境のハイパーメディア情報システムに最適な軽さと高速性をもつ

リケーション・レベルのプロトコルです。HTTPは、分散環境のハイパーメディア情報システムに最適な軽さと高速性を兼ね備えているとともに、各種マルチメディア・ツールと組み合わせることで、簡単にマルチメディアを実現することができます。このため、研究者や学術関係者のみならず、ビジネス、教育、医療、マスメディアといった、すそ野の広いインターネットの大ブームを引き起こす原動力になりました。

実用的な情報システムは、単純な情報取得だけではなく、検索、更新などのより多くの機能も必要とします。HTTPでは、リクエストの目的を示すメソッドを用いることで、これらの機能を実現しています。メソッドが適用されるリソースを示す方法には、URI（Uniform Resource Identifier）と呼ばれる参照規約があり、URIには場所を示すURL（Uniform Resource Locator）や名前を示すURN（Uniform Resource Name）があります。メッセージはインターネット・メールやMIME（4. 3. 4参照）で用いられるのと同様のフォーマットで通過します。

HTTPはまた、ユーザー・エージェント❶と、プロキシーやゲートウェイ間での、SMTP、NNTP、FTP、Gopher、WAISなどHTTP以外のプロトコルとの通信にも使用されます。これにより、分散アプリケーションとして利用可能なリソースへのハイパーメディア・アクセスを可能にしています。

❶ユーザー・エージェント（user agent）…リクエストを発行するクライアントのこと。ほとんどの場合はブラウザだが、エディタやスパイダー（Webをわたり歩くロボット）、またはそれ以外のエンド・ユーザー・ツールであることがある。

(2) HTTPで送られるデータ形式

HTTPでは基本的な特徴として、プロトコルそのものと、そのプロトコル上で送られるデータ形式を独立させています。

HTTPは、アンカーと呼ばれるタグによって他のリソースへのハイパーリンクを実現できる言語であるHTML（HyperText Markup Language）との組み合わせで広く世の中に普及しましたが、HTMLはHTTP上で送られるデータ形式の一つにすぎません。

三次元データの記述などインタラクティブなシミュレーションを行うための言語であるVRML（Virtual Reality Modeling Language）や専用のコンパイラで作成されたバイト・コードを送り、クライアントのそれぞれの環境でネイティブなマシン語に変換し実行するオブジェクト指向プログラミング言語のJava、また将来開発されるであろう言語も含めて、あらゆるデータをユーザー・エージェントと組み合わせることによって、インターネット上でのマルチメディアを実現しています。

(3) ブラウザ

ユーザー・エージェントは、一般にブラウザ（Browser）と呼ばれています。フリーソフトウェアも含めて各種のブラウザが利用可能です。

❷ HTTP/1.0の送受信の仕組み

HTTPは、リクエストとレスポンスの単純な仕組みによって成り立っています。

クライアントはサーバとコネクションを確立し、後述するリクエスト・メソッ

ド、URI、プロトコル・バージョン、MIMEライクなメッセージという形式で、サーバにリクエストを送ります。

この最後のメッセージは、リクエスト・モディファイヤ、クライアント情報、ボディ・コンテンツを含んでいます。これに対してサーバは、メッセージのプロトコル・バージョン、通信の成功や失敗の情報を伝えるステータス・コード、MIMEライクなメッセージを含んだステータス・ラインを返送します。この最後のメッセージは、サーバ情報、エンティティ・メタ情報、ボディ・コンテンツを含んでいます。

インターネットにおいて、HTTPの通信はTCP/IPのコネクション上で行われます。デフォルトのポートはTCPの80ですが、他のポートも使用可能です。このことで、インターネットの他のプロトコルや、インターネット以外の他のネットワーク上のプロトコルでは、HTTPは実装できないということはありません。HTTPは単純に信頼できるトランスポート層の使用を仮定しているだけです。

実験的なアプリケーションをのぞいて、個々のリクエストに先立ってクライアントによってコネクションが確立され、レスポンスが送られた後にサーバによってコネクションが終了されます。クライアントとサーバのどちらも、それぞれがコネクションを途中で切断するかもしれないということを、了解しておかなくてはなりません。その理由は、ユーザー・アクション、自動タイムアウト、プログラム・ミスなどです。一方または両方によるコネクションの終了があれば、どのような場合でもそのときのリクエストを中断させることになります。

(1) もっとも単純な送受信

大部分のHTTP通信は、ユーザー・エージェントによって開始され、オリジン・サーバ❶上のリソースがリクエストされます。もっとも単純な送受信の例(図10-27)は、ユーザー・エージェントとオリジン・サーバの間に中継者は存在せず、シングル・コネクションで、データが送受信されるケースです。

(2) 中継者が存在する送受信

HTTPのリクエストとレスポンスにおいて、一つ以上の中継者が存在し、複数のコネクションを通してデータが送受信される場合、状況は少し複雑になります。

図10-28において、ユーザー・エージェントとオリジン・サーバの間には、A、B、Cの三つの中継者が存在します。このリクエストとレスポンスの連鎖を通るメッセージは、四つのコネクションを経由しなくてはなりません。いくつかのHTTPの通信のオプションは、いちばん近くでトンネルではないサイトとのコネクションに適用されたり、連鎖の終端に適用されたり、または連鎖のすべてのコネクションに適用されます。

この図は直線的に書かれていますが、個々のパートは複数であったり、同時通信であるかもしれません。例えば、BはA以外の複数のクライアントからリクエストを受けとるかもしれないし、C以外のサーバにリクエストを転送するかもしれません。それと同時に、Aのリクエストを処理しているかもしれません。

❶オリジン・サーバ (origin server) …提供されるリソースが、存在するかまたはつくられる。サーバのこと。

ユーザー・エージェントとオリジン・サーバの間には、プロキシー、ゲートウェイ、トンネルといった3種類の中継者が存在することがあります（図10-29）。

① プロキシー（proxy）

プロキシーとは、他のクライアントに代わってリクエストを行う代理プログラムのことです。プロキシーはサーバとクライアントの両方の役割をします。

プロキシーはリクエストを転送する前に、リクエスト・メッセージを翻訳し、

ポイント 図10-27

❶ ユーザー・エージェントとオリジン・サーバだけの間の通信

❷ シングル・コネクションでリクエストとレスポンス

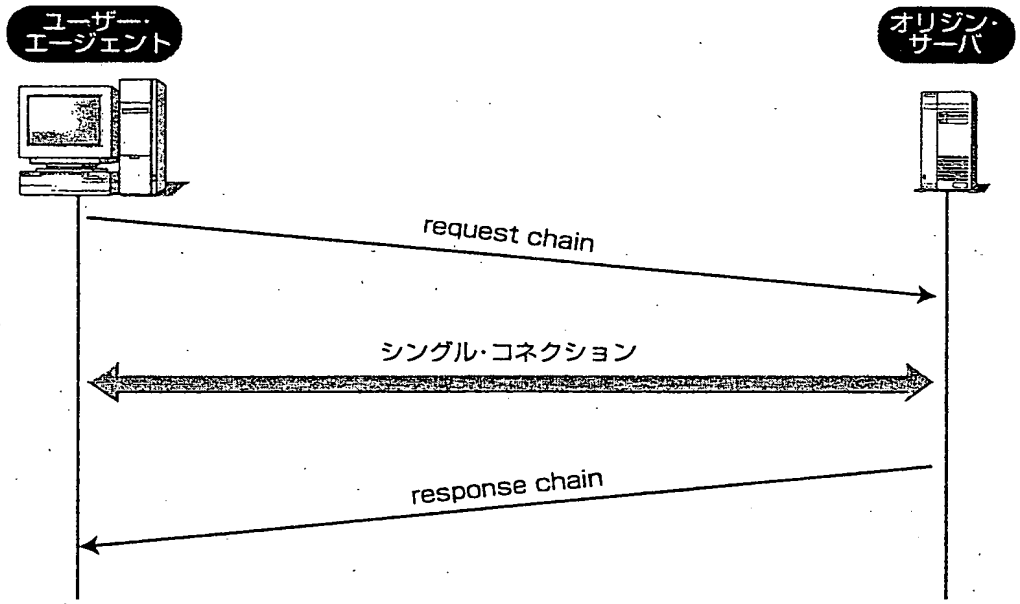


図10-27 もっとも単純な送受信の例

ポイント 図10-28

❶ ユーザー・エージェントとオリジン・サーバ間に一つ以上の中継者が介在する場合

中継者にはプロキシー、ゲートウェイ、トンネルがある

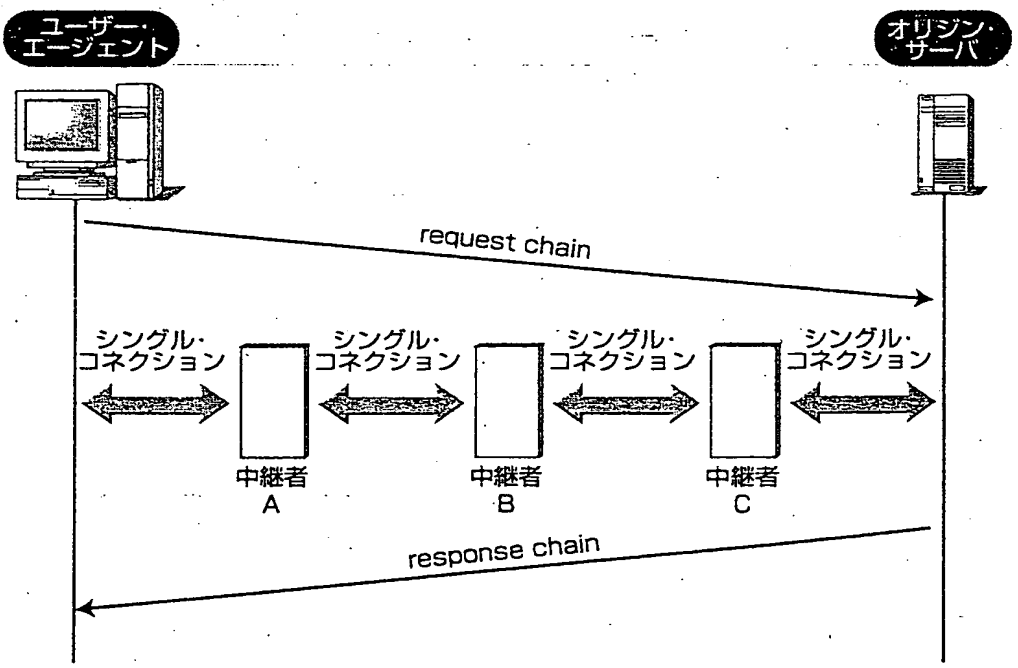


図10-28 中継者が存在する送受信の例

場合によってはそれを書き直します。ファイアウォールのクライアント側の入口としてしばしば使用され、ユーザー・エージェントに実装されていないプロトコルのリクエストを代理で行います。

② ゲートウェイ（gateway）

ゲートウェイは、他のサーバの中継ぎをするサーバです。プロキシーと異なり、リクエストしたクライアントから、あたかもオリジン・サーバとしてリクエストを受けとります。クライアントは、自分がゲートウェイと通信をしているかどうかはわかりません。ゲートウェイは、ネットワークのファイアウォールを通じたサーバ側の入口として、しばしば使用され、HTTP以外のサーバに蓄えられているリソースにアクセスするプロトコルの変換サーバとしても、使用されます。

③ トンネル（tunnel）

トンネルは、二つのコネクション間で目隠しリレーのような中継を行うプログラムです。ひとたびアクティブになったら、HTTPのリクエストから起動されたものにもかかわらず、HTTPのプログラムの一部としては働きません。トンネルは、コネクションのどちらかが切れたときに、プログラムが終了します。ファイアウォールはこれに相当します。

(3) キャッシュが介在する送受信

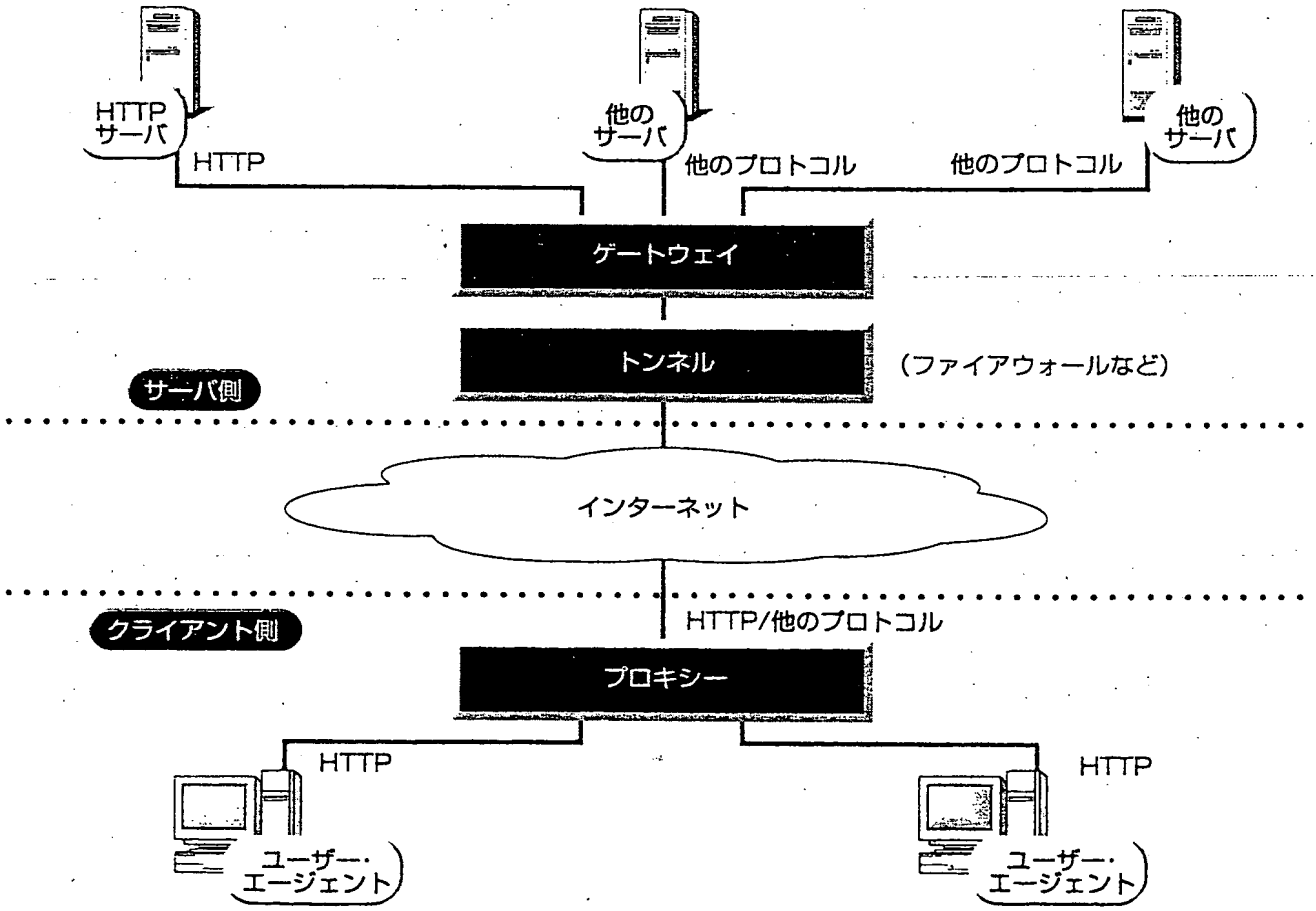


図10-29 プロキシー/ゲートウェイ/トンネル

キャッシュ（cache）…ローカルに蓄えられたレスポンス・メッセージまたはそのメッセージの蓄積、取得、削除を行うシステムのこと。将来同じリクエストがきたときに、レスポンス・タイムとネットワークの混雑を削減するために、キャッシュ可能なレスポンスを蓄えておく。どのようなクライアントやサーバもキャッシュをもつことができるが、サーバがトンネルとして働いているときはキャッシュを使うことができない。

トンネルとして中継を行わない通信パートでは、リクエスト処理においてキャッシュ 機能を用いることができます。リクエスト/レスポンスの連鎖のどこかのパートがそのリクエストのキャッシュされたレスポンスを保持していれば、この連鎖を短くすることができることが、キャッシュの特徴です。

図10-30は、ユーザー・エージェント自身やAにキャッシュされていないリクエストに対して、以前にオリジン・サーバからCを通して入手したレスポンスのキャッシュされたデータをBがもっていて、それが用いられるということを示しています。

❸ HTTPの表記法と基本ルール

HTTPにおける表記法の拡張BNF（Backus Naur Form、RFC822参照）を表10-12に、またHTTPの基本ルールを表10-13に示します。

❹ プロトコルのパラメータ

(1) HTTPのバージョン

HTTPでは、"〈major〉.〈minor〉" という書式をバージョン番号として用います。この目的は、送信者がその後に続くHTTP通信のメッセージ・フォーマットと容量を伝えることです。

通信形態に変更がないメッセージの追加や、単なる拡張可能なフィールドの値の追加では、バージョン番号は変わりません。「一般的なメッセージのパースのアルゴリズムに変更はないが、メッセージの意味を追加したり、送信者の追加機能を求める」という、プロトコルの機能に追加の変更があったときは、〈minor〉番号が一つ繰り上げられます。一方、メッセージ・フォーマットの変

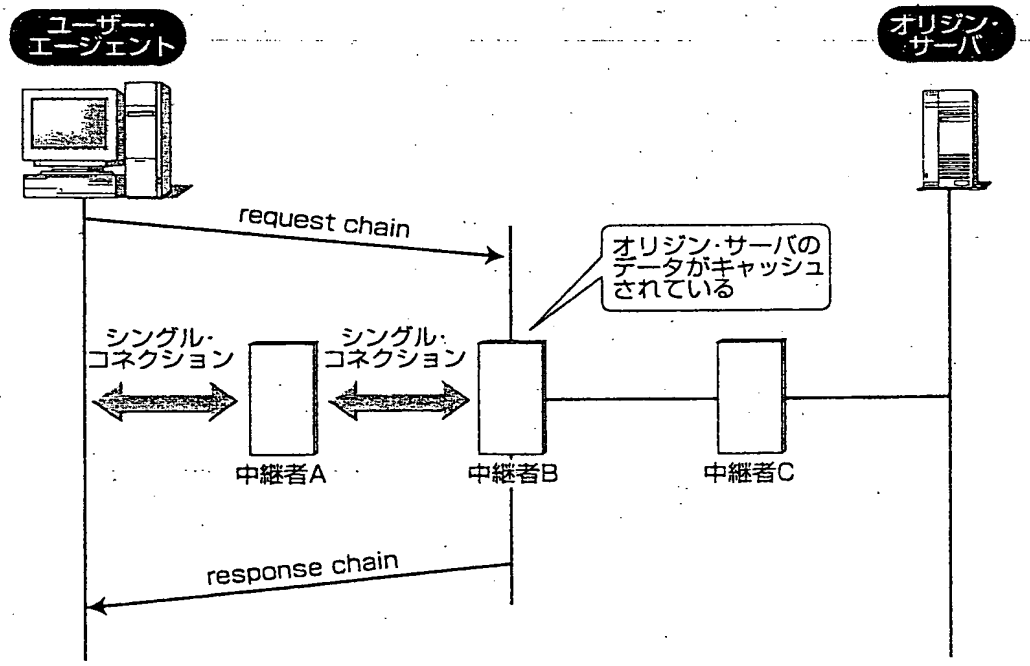


図10-30 キャッシュが介在する送受信

ポイント 図10-30

❶ キャッシュを用いてレスポンス・タイムとネットワークの混雑を削減

トンネルではキャッシュが使えない

更があったときには、〈major〉番号が一つ繰り上げられます。

HTTPメッセージのバージョン番号は、メッセージの最初の行のHTTP-Versionフィールドに書かれます。もしバージョン番号が省略されたら、HTTP/0.9のフォーマットであるとみなされます。

```
HTTP-Version= "HTTP" "/" 1*DIGIT "." 1*DIGIT
(HTTP/0.9 < HTTP/1.0 < HTTP/1.1という順番になります)
```

Full-RequestまたはFull-Responseのメッセージを送信するアプリケーションは、"HTTP/1.0" というHTTP-Versionフィールドをもたなくてはなりません。

HTTP/1.0サーバは、次の条件を満たす必要があります。

① HTTP/0.9とHTTP/1.0のリクエストのRequest-Lineの書式を認識すること

② HTTP/0.9かHTTP/1.0の書式のどのような有効なリクエストも理解すること

③ クライアントで使用されているプロトコル・バージョンと同じバージョンで、適当なメッセージを返送すること

またHTTP/1.0のクライアントは、次の条件を満たす必要があります。

① HTTP/1.0のレスポンスのStatus-Lineの書式を認識すること

② HTTP/0.9かHTTP/1.0の書式のいかなる有効なレスポンスも理解すること

プロキシーとゲートウェイのアプリケーションにおいて、そのアプリケーション自身のバージョンとは異なるフォーマットのリクエストを転送するときに

構　文	説　明
name=definition	あるルールの名前は、単純にその名前自身であり、"=" によってその定義と分離される
"literal"	ダブルクォーテーションで囲まれたテキストそのもの。断りがなければケース・インセンシティブ（大文字と小文字を区別しない）
rule1 | rule2	rule1かrule2のどちらか
(rule1　rule2)	かっこで囲まれた要素は、一つの要素として扱われる
*rule	要素の前の "*" は繰り返しを意味する。"<n>*<m>element" が完全形で、最少n回、最大m回、要素が現れる。デフォルトではゼロと無限大。"*(element)" はいくつでもよい
[rule]	オプショナル要素
N rule	反復回数を指定する
#rule	完全フォーマットは "<n>#<m>element" 。要素のリストを定義する

表10-12　HTTPの表記法

は、注意が必要です。プロトコルのバージョンは送信側のプロトコルの能力を示しているので、プロキシーやゲートウェイは、自分自身のバージョンより高いバージョンのものを転送することはできません。もしプロキシーやゲートウェイが、自分自身より高いバージョンのリクエストを受けとったならば、リクエストのバージョンを下げるか、またはエラーを返さなくてはなりません。反対に、自分より低いバージョンのリクエストを受けとったときには、バージョ

HTTP基本ルール	
OCTET	= <any 8-bit sequence of data>
CHAR	= <any US-ASCII character (octets 0-127)>
UPALPHA	= <any US-ASCII uppercase letter "A" .. "Z">
LOALPHA	= <any US-ASCII lowercase letter "a" .. "z">
ALPHA	= UPALPHA | LOALPHA
DIGIT	= <any US-ASCII digit "0" .. "9">
CTL	= <any US-ASCII control character (octets0-31) and DEL (127)>
CR	= <US-ASCII CR,carriage return (13)>
LF	= <US-ASCII LF,linefeed (10)>
SP	= <US-ASCII SP,space (32)>
HT	= <US-ASCII HT,horizontal-tab (9)>
<">	= <US-ASCII double-quote mark (34)>
CRLF	= CR LF
LWS	= [CRLF] 1*(SP | HT)
TEXT	= <any OCTET except CTLs,but including LWS>
HEX	= "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f" | DIGIT
word	= token | quoted-string
token	=1*<any CHAR except CTLs or tspecials>
tspecials	= "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" | "¥" | <"> | "/" | "[" | "]" | "?" | "=" | "{" | "}" | SP | HT
comment	= "(" *(ctext | comment) ")"
ctext	= <any TEXT excluding "(" and ")">
quoted-string	= (<">*(qdtext) <">)
qdtext	= <any CHAR except <"> and CTLs,but including LWS>

表10-13　HTTPの基本ルール

ンを上げてから転送することになります（図10-31）。

(2) URI (Uniform Resource Identifier)

① URIの概要

URI（Uniform Resource Identifier、ユニフォーム・リソース・アイデンティファイヤ）は、インターネットのオブジェクトのアドレスや名前をエンコードした、WWWで用いられる書式です。

WWWには、WWWのために開発された既存のプロトコルや、将来開発されるであろうプロトコルなど、拡張の可能性のある多くのプロトコルを用いてアクセスされるオブジェクトが存在します。あるプロトコルにおける個々のオブジェクトへのアクセス方法は、アドレスの文字列の形式にエンコードされています。しかし他のプロトコルでは、いろいろな形式のオブジェクトの名前の利用が可能です。そこでWWWでは、汎用的なオブジェクトの概念を抽象化するため、オブジェクトの名前やアドレスの統一的なセットのコンセプトが必要になります。

URIは、登録されたネーム・スペースやアドレスにおける名前のこの統一的なセットで、次のようないろいろな名前で知られています。

(a) WWWアドレス（WWW address）

(b) ユニバーサル・ドキュメント・アイデンティファイヤ（Universal Document Identifier）

(c) ユニバーサル・リソース・アイデンティファイヤ（Universal Resource Identifier）

(d) URL（Uniform Resource Locator、ユニフォーム・リソース・ロケータ）とURN（Uniform Resource Name、ユニフォーム・リソース・ネーム）の組み合わせ

URLは、ネットワークのプロトコルを使用したアクセス・アルゴリズム上へマッピングしたアドレスを表現する、URIの一つの形式です。既存のインターネット・アクセス・プロトコルでは、ほとんどの場合、アクセス・アルゴリズムの

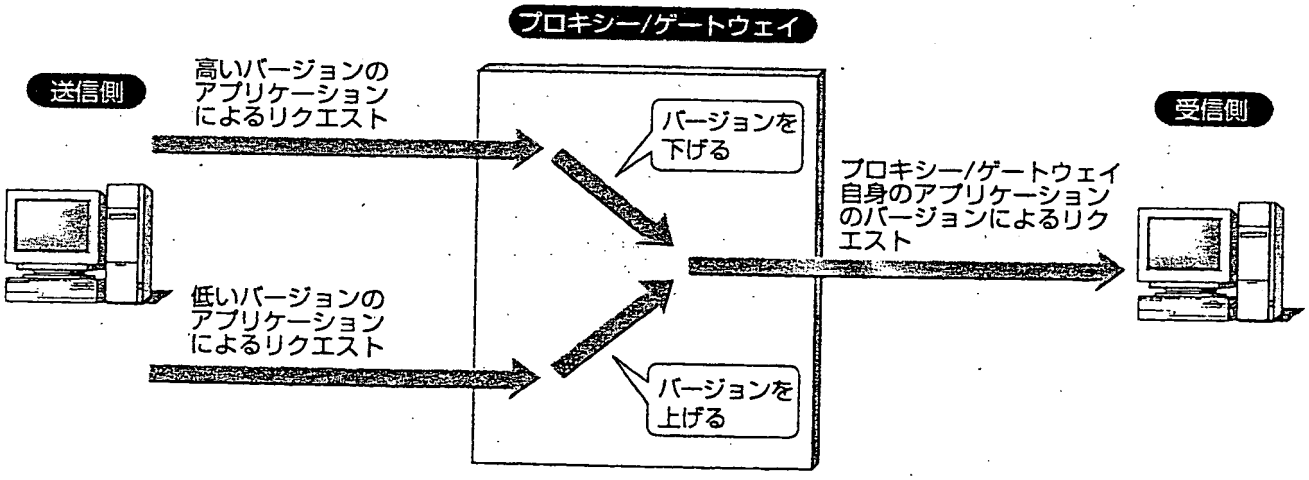


図10-31 プロキシーやゲートウェイにおけるプロトコル・バージョンの処理

エンコーディングを、アドレスを示す何か簡単なものに定義する必要があります。既存のプロトコルでアクセスされるオブジェクトを参照するURIは、URLとして知られています。

② 一般文法

HTTPでのURIは、絶対フォームと相対フォームのどちらかで記述されます。このフォームの違いは、絶対フォームのURIでは、スキーム（Scheme）の名前とコロン（：）ではじまる点です（表10-14）。

なお、スキームの具体例としては表10-15に示すようなものがあります。

③ HTTPのURL

“http”のスキームは、HTTPのプロトコルを介したネットワーク・リソース

HTTP一般文法	
URI	= (absoluteURI | relativeURI) ["#" fragment]
absoluteURI	= scheme ":" * (uchar | reserved)
relativeURI	= net_path | abs_path | rel_path
net_path	= "//" net_loc [abs_path]
abs_path	= "//" rel_path
rel_path	= [path] [";" params] ["?" query]
path	= fsegment* ("/" segment)
fsegment	= 1*pchar
segment	= *pchar
params	= param* (";" param)
param	= * (pchar | "/")
scheme	= 1* (ALPHA | DIGIT | "+" | "-" | ".")
net_loc	= * (pchar | ";" | "?")
query	= * (uchar | reserved)
fragment	= * (uchar | reserved)
pchar	= uchar | ":" | "@" | "&" | "=" | "+")
uchar	= unreserved | escape
unreserved	= ALPHA | DIGIT | safe | extra | national
escape	= "%" HEX HEX
reserved	= ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+"
extra	= "!" | "*" | "'" | "(" | ")" | ","
safe	= "$" | "-" | "_" | "."
unsafe	= CTL | SP | <"> | "#" | "%" | "<" | ">"
national	= <ALPHA,DIGIT,reserved,extra,safe,unsafeをのぞくOCTET>

表10-14 HTTPのURIの一般文法

を指定するときに用いられます。

```
http_URL        = "http：" "//" host [ " ：" port] [abs_path]
host            = 〈インターネット上のホスト・ドメイン・ネームか
                  IPアドレス（ドットで区分された10進数)〉
port            = *DIGIT
```

この意味は、指定されたリソースは、ホストのTCPのそのポート番号を見ているサーバ上に存在し、リソースのRequest-URLはabs_pathに書かれているということです。

ここで [" ：" port] が省略された場合は、80が指定されます。また空のabs_pathは、"/" に置き換えられます。ホスト名は大文字と小文字の区別がなく（ケース・インセンシティブ❶)、大文字のホスト名は小文字に変換されます。

HTTPはトランスポート層プロトコルとは独立したものですが、HTTPのURLは、TCPだけを用いてリソースを指定します。したがって、非TCPのリソースについては、他のURIスキームを用いなくてはなりません。

❶…これに対して大文字と小文字を区別することを「ケース・センシティブ」という。

④ 日時のフォーマット

HTTP/1.0のアプリケーションでは、次の三つのフォーマットを用いて日時を表します。

```
Sun, 06 Nov 1994 08：49：37 GMT     ; RFC 822, updated by RFC 1123
Sunday, 06-Nov-94 08：49：37 GMT    ; RFC 850, obsoleted by RFC 1036
Sun Nov6 08：49：37 1994            ; ANSI Cのasctime () フォーマット
```

ここでは、最初のフォーマットがインターネット・スタンダードとして望まし

スキーム	内　容
ftp	File Transfer Protocol
http	HyperText Transfer Protocol
gopher	Gopher Protocol
mailto	Electronic-mail Address
news	USENET (NetNews) News
nntp	USENET (NetNews) News Using NNTP Access
telnet	Telnet
wais	Wide Area Information Servers
file	Host-specific File Names
prospero	Prospero Directory Service

表10-15　スキームの例

く、RFC 1123で定められたフォーマットの固定長のサブセットになっています。2番目のフォーマットはしばしば使用されますが、すでに古くなったRFC 850のデータ・フォーマットをベースにしており、年号が2桁です。この日付の値をパースするHTTP/1.0のクライアントとサーバの両方とも、3番目のフォーマットを発行することはありませんが、この三つすべてのフォーマットを受理しなくてはなりません。

すべてのHTTP/1.0のdate/timeスタンプは、GMT（Greenwich Mean Time、グリニッジ標準時）として知られるユニバーサル・タイム（Universal Time）で表現されなくてはなりません。

最初の二つのフォーマットの"GMT"という省略形が、それを表しており、asctimeのフォーマットでもGMTを前提としています。

```
HTTP-date     = rfc1123-date | rfc850-date | asctime-date
rfc1123-date  = wkday "," SP date1 SP time SP "GMT"
rfc850-date   = weekday "," SP date2 SP time SP "GMT"
asctime-date  = wkday SP date3 SP time SP 4DIGIT
```

⑤ キャラクタ・セット

HTTPでは、MIMEと同様のキャラクタ・セット(Character Sets)を用います。

```
charset = "US-ASCII"
        | "ISO-8859-1" | "ISO-8859-2" | "ISO-8859-3"
        | "ISO-8859-4" | "ISO-8859-5" | "ISO-8859-6"
        | "ISO-8859-7" | "ISO-8859-8" | "ISO-8859-9"
        | "ISO-2022-JP" | "ISO-2022-JP-2" | "ISO-2022-KR"
        | "UNICODE-1-1" | "UNICODE-1-1-UTF-7"
        | "UNICODE-1-1-UTF-8" | token
```

キャラクタ・セットは、ケース・インセンシティブなトークンで識別されます。

⑥ コンテント・コーディング

リソースに適用されたエンコード方法を示すために、Content Coding値を使用します。コンテント・コーディング（Content Coding）は、もとのメディア・タイプを失わずにドキュメントを圧縮したり暗号化するために用いられます。

```
content-coding= "x-gzip" | "x-compress" | token
```

すべてケース・インセンシティブです。Content-Encoding（表10-18）ヘッダ・フィールドの中で、content-coding値を使用します。ここで重要なことは、どのようなデコードを行うことが必要かを示しているということです。

⑤ HTTPメッセージ

(1) メッセージ・タイプ

HTTPメッセージは、クライアントからサーバへのリクエストと、サーバからクライアントへのレスポンスの二つから構成されています。

```
HTTP-message    = Simple-Request          ; HTTP/0.9メッセージ
                | Simple-Response
                | Full-Request            ; HTTP/1.0メッセージ
                | Full-Response
```

Full-RequestとFull-Responseはエンティティを送るため、RFC 822の汎用メッセージ・フォーマットを使用します。どちらのメッセージも、CRLF以外何もない1行の空白行で分割された、オプショナルのヘッダ・フィールドと、Entity-Bodyを含みます。

```
Full-Request    = Request-Line            ; (P.553参照)
                  *(General-Header        ; (P.552参照)
                  | Request-Header        ; (P.554参照)
                  | Entity-Header)        ; (P.559参照)
                  CRLF
                  [Entity-Body]           ; (P.560参照)
Full-Response   = Status-Line             ; (P.555参照)
                  *(General-Header        ; (P.552参照)
                  | Response-Header       ; (P.554参照)
                  | Entity-Header)        ; (P.559参照)
                  CRLF
                  [Entity-Body]           ; (P.560参照)
```

Simple-RequestとSimple-Responseにはヘッダ情報がなく、GETという一つのリクエスト方法（メソッド）に限定されています。

```
Simple-Request  = "GET" SP Request-URI CRLF
Simple-Response = [Entity-Body]
```

(2) メッセージ・ヘッダ

General-Header（P.552参照）、Request-Header（P.554参照）、Response-Header（P.554参照）、Entity-Header（10. 7. 1）のフィールドを含む、HTTPのヘッダ・フィールドは、RFC 822に記述されているものと同じ汎用フォーマットにしたがいます。個々のヘッダ・フィールドは、名前、コロン（":"）、一つ

のスペース（SP）、フィールド値から構成されます。

フィールド・ネームはケース・インセンシティブです。ヘッダ・フィールドは、行の先頭に少なくとも一つのSPかHT（タブ）をつけて、複数行に拡張することができますが、推奨はされません。

```
HTTP-header     = field-name ":" [field-value] CRLF
field-name      = token
field-value     = *(field-content | LWS)
field-content   = 〈*TEXTかtoken、tspecials、quoted-stringの組み
                    合わせで構成される、フィールド値を表すOCTET〉
```

ヘッダ・フィールドの順番は重要ではありませんが、General-Headerフィールドを最初に送り、Request-HeaderかResponse-Headerフィールドが続き、Entity-Headerフィールドに移るのが一般的です。

(3) ゼネラル・ヘッダ・フィールド

いくつかのゼネラル・ヘッダ・フィールド（General Header Field）は、リクエストとレスポンスのメッセージの両方に適用可能ですが、エンティティには適用されないものがあります。それらのヘッダは、送信されるメッセージだけに適用されます。

```
General-Header  = Date                  ; (表10-18参照)
                | Pragma                ; (表10-18参照)
```

⑥ リクエスト

クライアントからサーバに対するリクエストでは、メッセージの最初の行で、そのリソースに適用されるメソッド、リソースの識別子、使用されるプロトコルのバージョンが記述されます。機能が限定されたHTTP/0.9のプロトコルに対する下位互換としては、次の二つの書式が有効です。

```
Request         = Simple-Request | Full-Request
Simple-Request  = "GET" SP Request-URI CRLF
Full-Request    = Request-Line                  ; (P.552参照)
                  *(General-Header              ; (P.551参照)
                   | Request-Header             ; (P.554参照)
                   | Entity-Header)             ; (P.559参照)
                  CRLF
                  [Entity-Body]                 ; (P.560参照)
```

もしHTTP/1.0のサーバがSimple-Requestを受けとったら、そのサーバは

HTTP/0.9のSimple-Responseを返さなくてはなりません。Full-Responseを受けとる能力のあるHTTP/1.0のクライアントは、Simple-Requestを発行してはいけません。

(1) Request-Line

Request-Lineはメソッドのトークンではじまり、その後にRequest-URIとプロトコルのバージョンが続き、CRLFで終わります。個々の要素は、スペース(SP)で区切られます。最後のCRLF以外では、CRやLFは使用できません。

```
Request-Line    =Method SP Request-URI SP HTTP-Version CRLF
```

Simple-RequestとFull-RequestでのRequest-Lineの違いは、HTTP-Versionフィールドの存在と、GET以外のメソッドが許されるかどうかです。

① メソッド (Method)

メソッドは、Request-URIで指定されるリソースに対して実行される方法を示しています。メソッドのトークンは、ケース・センシティブです。

```
Method          = "GET"                    ; (P.561参照)
                | "HEAD"                   ; (P.562参照)
                | "POST"                   ; (P.563参照)
                | extension-method
extension-method=token
```

あるメソッドがあるリソースに対して許可されていなかった場合、クライアントはそのレスポンスに含まれるステータス・コードで、それを認識します。一方サーバは、もしそのメソッドを認識できなかったり実装されていなかった場合は、ステータス・コード501 (Not Implemented) を返さなくてはなりません。

② Request-URI

Request-URIはUniform Resource Identifierであり、リクエストが適用されるリソースを特定するものです。

```
Request-URI    =absoluteURI | abs_path
```

この二つのオプションは、リクエストの性格そのものに依存しています。absoluteURIはプロキシーに対してリクエストが行われるときにだけ許されます。プロキシーはリクエストを転送し、レスポンスを返すよう要求されます。もしメソッドがGETかHEADで、前のレスポンスがキャッシュされている場合は、Expiresヘッダ・フィールドの条件をパスするならば、プロキシーはキャッシュ・データを使用することができます。

ここでそのプロキシーは、別のプロキシーへリクエストしたり、直接absoluteURIで示されるサーバに転送することができます。リクエストがループ状

態になるのを避けるため、プロキシーは、すべてのエイリアス、ローカル変数、IPアドレスを含む、そのサーバのすべての名前を把握できなくてはなりません。

例えば、次のようなRequest-Lineを仮定します。

```
GET  http：//www. w3. org/pub/WWW/TheProject. html  HTTP/1.0
```

Request-URIのもっとも一般的な書式は、オリジン・サーバやゲートウェイ上のリソースを特定するために用いられます。この場合、URIの絶対パス（abs_path）だけが送られます。例えば、オリジン・サーバにあるリソースを取得したいクライアントは、ポート80のTCPコネクションをホストwww.w3.orgに張り、次の行を送ります。

```
GET  /pub/WWW/TheProject. html  HTTP/1.0
```

この後に、Full-Requestの残りの部分を送ります。ここで、絶対パス（abs_path）は空白であってはならないことに注意する必要があります。はじめのURIがない場合には、"/"（サーバのルート）を送らなくてはなりません。

Request-URIは、RFC1738で定義されたエンコード方法の"%HEX HEX"を用いていくつかのキャラクタをエスケープする、エンコード文字列として送られます。オリジン・サーバは、このリクエストを正しく解釈するため、Request-URIをデコードする必要があります。

(2) Request-Headerフィールド

クライアントは、Request-Headerフィールドを用いて、リクエストやクライアント自身に関する追加情報をサーバに伝えます。

```
Request-Header = Authorization          ;（表10-18参照）
               | From                   ;（表10-18参照）
               | If-Modified-Since      ;（表10-18参照）
               | Referer                ;（表10-18参照）
               | User-Agent             ;（表10-18参照）
```

❼ レスポンス

リクエスト・メッセージを受信したり中断したりしたときは、サーバは次のようなHTTPレスポンス・メッセージの形式で返信します。

```
Response        = Simple-Response | Full-Response
Simple-Response =[Entity-Body]
Full-Response   = Status-Line                    ; (P.555参照)
                  *(General-Header               ; (P.552参照)
                   | Response-Header             ; (P.556参照)
                   | Entity-Header)              ; (P.559参照)
                   CRLF
                   [Entity-Body]                 ; (P.560参照)
```

ここでSimple-Responseは、HTTP/0.9のSimple-Requestに対するレスポンスとしてか、機能が限定されるHTTP/0.9しかサーバがサポートしていないときにだけ送られます。もしクライアントがHTTP/1.0のFull-Requestを送って、Status-Lineではじまらないレスポンスを受けとるならば、そのレスポンスはSimple-Responseであり、そのように処理しなくてはなりません。Simple-ResponseはEntity-Bodyだけで構成されていて、サーバがコネクションを切断することによって終了します。

(1) ステータス・ライン (Status-Line)

Full-Responseメッセージの最初の行のStatus-Lineは、プロトコル・バージョン、ステータス・コード、それに相当する理由文からなり、それぞれはスペース(SP) で分けられています。いちばん最後のCRLFをのぞいて、CRやLFは使われません。

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

ステータス・ラインは、必ずプロトコル・バージョンとステータス・コードではじまるので、Simple-ResponseとFull-Responseを区別することができます。

① ステータス・コードと理由文

ステータス・コード (Status-Code) は、そのリクエストを解釈し返信しようとした結果を示す3桁の数字です。理由文 (Reason-Phrase) はステータス・コードのテキストによる記述です。ステータス・コードはソフトウェアに対して、理由文はユーザーに対して、それぞれ情報を提供します。クライアントは、理由文を調べたり表示することは要求されません。

ステータス・コードの百の桁はレスポンスのクラスを示し1～5まであります。十と一の桁は、何らのカテゴリも意味しません (表10-16)。

HTTP/1.0で定義される個々のステータス・コードの値と、対応する理由文は、表10-17に示す通りです。ここに掲載した理由文はあくまでも推奨で、ローカルに等しいものに置き換えることができます。

HTTPのステータス・コードは拡張可能で、表10-17にあげたコードは現在の実装で一般に認識されているものだけです。HTTPのアプリケーションが、す

べての登録されたステータス・コードの意味を知ることは、望ましいことですが、要求されているわけではありません。その代わり、百の桁が示すステータス・コードのクラスは必ず理解して、そのX00番と同じステータスであるとして、処理しなくてはなりません。その場合の例外として、401（Unauthorized）のレスポンスはキャッシュしてはなりません。

例えば、450という意味のわからないステータス・コードをクライアントが受信した場合には、あたかも400を受けとったかのようにレスポンスを扱い、リクエストに何らかの問題があったと認識するのが安全です。そのような場合には、ユーザー・エージェントはレスポンスとともに返ってくるエンティティを表示するべきです。そのエンティティが、人間が理解できる形式で障害情報をもっているからです。

② Response-Headerフィールド

サーバは、Response-Headerフィールドに、ステータス・ラインに書けないレスポンスに関する追加情報を書くことができます。サーバに関する情報や、Request-URIで示されるリソースへのアクセスに関する情報を提供します。

百の桁	クラス	内　容
100番台	情報	Status-Lineとオプショナル・ヘッダだけからなり、空の行で終了している、仮のレスポンスです。HTTP/1.0では、100番台のステータス・コードは定義しません。またHTTP/1.0のリクエストに対しては、このレスポンスは無効です。これは、規格外の実験的なアプリケーションで使用されることがあります。
200番台	成功	クライアントのリクエストは受信に成功、解釈、受諾されたということを示しています。
300番台	リダイレクション	リクエストを満足させるためには、ユーザー・エージェントによるさらなるアクションが必要であることを示しています。メソッドがGETかHEADである場合、またはそうであるときだけ、ユーザーの操作なしにユーザー・エージェントによって必要なアクションが行われます。無限ループを防ぐため、ユーザー・エージェントは、自動的に5回以上リクエストをリダイレクトしてはいけないことになっています。
400番台	クライアント・エラー	クライアントがエラーを起こした可能性を示しています。もしクライアントが400番台のステータス・コードを受けとったときにまだリクエストを終了していなかったときは、すぐにサーバに対してデータの送信を中止するべきです。HEADリクエストにレスポンスするときをのぞいて、状況が一時的であれ永続的であれ、サーバはエラーの状況の説明をエンティティに含めなくてはなりません。これらのステータス・コードは、どのようなリクエスト・メソッドにも適用されます。
500番台	サーバ・エラー	サーバが自分のエラーを感知したか、そのリクエストを実行できないときのものです。クライアントがこのコードを受けとったときにリクエストを終了していなければ、サーバにデータを送るのをすぐに中止しなくてはなりません。HEADのリクエストによるレスポンスをのぞき、サーバはエラーの状況の説明と、それが一時的なものかそうでないかを、エンティティに含めなくてはなりません。

表10-16　ステータス・コードの百の桁によるクラスの区分

ステータス・コード	理由文	説　明
200	OK	リクエストが成功したことを示しています。このレスポンスで返ってくる情報は、リクエスト時のメソッドに依存しています。 GET：要求されたリソースにかかわるエンティティは、レスポンスの中で送られる HEAD：レスポンスはヘッダ情報を含むが、エンティティ・ボディは含まない POST：アクションの結果を含んでいるエンティティ
201	Created	リクエストは処理され、新しいリソースが作成されることになった、ということを示しています。この新しくつくられたリソースは、レスポンスのエンティティで返されるURIから参照することができます。オリジン・サーバはこのStatus-Codeを使用する前にそのリソースを作成しなくてはなりません。もしこのアクションがすぐに実行されない場合には、サーバはレスポンス・ボディの中に、そのリソースがいつ利用可能になるのか記述しなくてはなりません。そうしない場合には、サーバは202（Accepted）を返す必要があります。 この仕様で定義されるメソッドでは、POSTだけがリソースをつくることができます。
202	Accepted	リクエストはサーバで処理するために受諾されたが、まだその処理が終了していないというものです。 この機能を用いることで、サーバでのプロセスが終了するまでコネクションを保ち続けることなしに、1日に1回実行するようなバッチ処理などの他のプロセスを、クライアントから実行することができるようになります。
204	No Content	サーバはリクエストを実行したが、新規に返送するデータはない、というものです。
300	Multiple Choices	要求されたリソースは一つ以上の場所で利用可能である、というものです。HEADのリクエストでなければ、レスポンスのエンティティの中に、ユーザーやユーザー・エージェントが選ぶことができるリソースのリストが含まれています。サーバが正しいリストを返してくれれば、Locationフィールドにそのurlが含まれており、自動的なリダイレクションに使用することができます。
301	Moved Permanently	要求されたリソースは新しいURLに移り、今後のこのリソースへのリクエストはそのURLを使用するべきである、というものです。リンク情報の編集能力のあるクライアントは、もし可能であれば、サーバから返される新しいURLに自動的にリンクしなおすべきです。
302	Moved Temporarily	要求されたリソースは、一時的に他のURLに移った、ということです。
304	Not Modified	もし、クライアントが条件つきGETのリクエストを行い、アクセスを許可されたが、そのリソースはIf-Modified Sinceフィールドに示された日時以降、変更はなかったという場合には、サーバはエンティティ・ボディなしでこのコードを送らなくてはなりません。
400	Bad Request	文法が間違っていたため、サーバはリクエストを解釈できなかった、ということです。クライアントは、そのままではリクエストを繰り返すことはできません。
401	Unauthorized	そのリクエストはユーザー認証を必要とする、ということです。要求されたリソースに対する適切な問いかけを含むWWW-Authenticateフィールドを、レスポンスはもたなくてはなりません。クライアントは適当なAuthorizationフィールドで、リクエストを繰り返すことになります。もしリクエストが、認証の信任状をはじめから含んでいるなら、このコードはその信任状が拒絶されたことを示しています。もし、401のレスポンスがその前のレスポンスと同じ問いかけを含んでいる場合で、ユーザー・エージェントがすでに少なくとも一度は認証を試みている場合には、ユーザーはエンティティを受けとることになります。
403	Forbidden	サーバはリクエストを理解したが、実行を拒否する、ということです。認証とは関係なく、リクエストも繰り返してはなりません。メソッドがHEADで、なぜ拒否したのかをサーバが知らせたいときは、エンティティ・ボディにその理由を記述します。
404	Not Found	サーバはRequest-URIに該当するものを何も見つけられなかった、ということです。それが一時的なものか永続的なものか、ということは示されません。もしサーバが、この情報をクライアントに知らせたくないのであれば、403（Forbidden）を代わりに用いるべきです。
500	Internal Server Error	サーバは、リクエストを実行できない予期しない状況に遭遇した、ということです。
501	Not Implemented	サーバは、リクエストを実行するために必要な機能をサポートしていない、ということです。これは、サーバがリクエスト・メソッドを認識できないときや、リソースに対してそれをサポートしていないときの、レスポンスです。
502	Bad Gateway	サーバは、ゲートウェイやプロキシーとして振る舞っているときに、リクエストを実行しようとしている上流のサーバから、無効なレスポンスを受けとった、ということです。
503	Service Unavailable	サーバは、一時的な過負荷やメンテナンスのために、今はリクエストを処理できない状況である、ということです。

表10-17　ステータス・コード一覧

フィールド名		内　容
Allow		Entity-HeaderのAllowフィールドは、Request-URIで指定されるリソースがサポートしているメソッドのセットを、リストで表しています。このフィールドの目的は、受信者にそのリソースに関して有効なメソッドを知らせることです。Allowは、POSTメソッドを使用したリクエストで使用することはできず、もしPOST形式の一部として送られても、無視されます。
	書式	Allow= "Allow" ":" 1#method
	例	Allow：GET,HEAD
Authorization		サーバに自分を認証してもらいたいユーザー・エージェントは、通常401のレスポンスを受けとった後に、リクエストの中にAuthorizationフィールドを含むことで、それを実行しようとします。このフィールドの値は、リソースの保護領域に入るための、ユーザー・エージェントの認証情報を含んだ信任状から構成されています。 リクエストが認証された場合は、同じ信任状はその領域内の他のすべてのリクエストに対して有効でなくてはなりません。認証フィールドを含むリクエストに対するレスポンスは、キャッシュすることはできません。
	書式	Authorization= "Authorization" ":" credentials
	例	Authorization：Basic SGlyYWtlIEdvbWEgR29tYQ==
Content-Encoding		Entity-HeaderのContent-Encodingフィールドは、media-typeのモディファイヤとして使用されます。これがある場合は、どのような追加のContent-codingがリソースに適用されるか、そしてContent-typeのヘッダ・フィールドで示されるmedia-typeを得るためにどのようなデコードを施さなくてはならないかを示しています。Content-Encodingは主に、ドキュメントを圧縮するときに使用されます。
	書式	Content-Encoding= "Content-Encoding" ":" Content-coding
	例	Content-Encoding：x-gzip
Content-Length		Entity-HeaderのContent-Lengthフィールドは、受信者に送られるEntity-Bodyのサイズを、10進のバイトで表しています。HEADメソッドの場合には、そのリクエストがGETメソッドで行われたときに送られるEntity-Bodyのサイズを表しています。
	書式	Content-Length= "Content-Length" ":" 1＊DIGIT
	例	Content-Length：3495
Content-Type		Entity-HeaderのContent-Typeフィールドは、受信者に送られるEntity-Bodyのメディア・タイプを示しています。HEADメソッドの場合のメディアのタイプは、そのリクエストがGETメソッドで行われたときに送られるメディア・タイプを表しています。
	書式	Content-Type= "Content-Type" ":" media-type
	例	Content-Type：text/html
Date		DateのGeneral-Headerフィールドは、RFC 822のorig-dateと同じ意味をもち、メッセージが発行された日時を表します。
	書式	Date= "Date" ":" HTTP-date
	例	Date：Tue,15 Jul 1961 08：12：31 GMT
Expires		Entity-HeaderのExpiresフィールドは、含まれるエンティティはその日時で有効期限が切れることを意味しています。アプリケーションは、その日を過ぎてエンティティをキャッシュしてはいけません。Expiresフィールドがあること自体は、その日時を境に、オリジナルのリソースが変更されたり、なくなってしまうことを意味するものではありませんが、リソースがある時期に変更されるとわかっているインフォメーション・プロバイダは、このExpiresフィールドを入れなくてはなりません。
	書式	Expires= "Expires" ":" HTTP-date
	例	Expires：Mon,01 Jan 2001 00：00：01 GMT
From		Request-HeaderのFromフィールドは、それがある場合は、ユーザー・エージェントを操作する人の電子メール・アドレスを示しています。
	書式	From= "From" ":" mailbox
	例	From：hiro@tky053.tth.expo96.ad.jp
If-Modified-Since		Request-HeaderのIf-Modified-Sinceフィールドは、GETメソッドとともに用いられ、もし要求されたリソースがこのフィールドで示された日時以降変更されていないのであれば、サーバからはリソースのコピーが返されないことを示しています。そのときは、304（Not Modified）がEntity-Bodyなしで返されます。
	書式	If-Modified-Since= "If-Modified-Since" ":" HTTP-date
	例	If-Modified-Since：Sat,29 Oct 1996 19：43：31 GMT

表10-18①　HTTP/1.0のヘッダ・フィールドの書式一覧

```
Response-Header = Location                    ; (表10-18参照)
                | Server                      ; (表10-18参照)
                | WWW-Authenticate            ; (表10-18参照)
```

⑧ エンティティ

Full-RequestやFull-Responseのメッセージは、リクエストやレスポンスの中でエンティティを転送します。エンティティは、Entity-Headerフィールドや、通常、Entity-Bodyで構成されます。

(1) Entity-Headerフィールド

フィールド名		内　容
Last-Modified		Entity-HeaderのLast-Modifiedフィールドは、リソースが最後に修正されたサーバが認識している日時を示しています。もし受信者がこのフィールドの日時よりも前のレスポンスのコピーをもっているならば、そのコピーは古いとみなされなければなりません。
	書式	Last-Modified= "Last-Modified" ":" HTTP-date
	例	Last-Modified：Tue,15 Nov 1996 12：45：26 GMT
Location		Response-HeaderのLocationフィールドは、Request-URIで示されるリソースの正確なLocation（場所）を示します。例えば3xxのレスポンスでは、このLocationはリソースの自動的なリダイレクションのための、望ましいURLを示さなくてはなりません。ここでは、absolute URLだけが使用されます。
	書式	Location= "Location" ":" absolute URI
	例	Location：http：//tky053.tth.expo96.ad.jp/
Pragma		General-HeaderのPragmaフィールドには、リクエストとレスポンスの連鎖において、どの受信者にも適用される実装に依存した指示が入ります。 もし、"no-cache" の指示がリクエスト・メッセージの中で現れたら、リクエストされているものがキャッシュされていても、アプリケーションはそのリクエストをオリジン・サーバに転送しなくてはなりません。これによってクライアントは、キャッシュされている古いデータをリフレッシュすることができます。
	書式	Pragma= "Pragma" ":" 1＃pragma-directive pragma-directive=：＂no-cache" ｜ extension-pragma extension-pragma=token ｜ "=" word｝
Referer		Request-HeaderのRefererフィールドによって、クライアントはサーバのために、Request-URIを取得したところのリソースのアドレス（URI）を示します。これによってサーバは、参考、ログ、キャッシュの最適化などのための、リソースへのバックリンクのリストを作成することができます。
	書式	Referer= "Referer" ":" (absoluteURI ｜ relativeURI)
Server		Response-HeaderのServerフィールドは、リクエストを処理するためにオリジン・サーバで使用されているソフトウェアの情報を含んでいます。
	書式	Server= "Server" ":" 1＊ (product ｜ comment)
	例	Server：CERN/3.0 libwww/2.17
User-Agent		Request-HeaderのUser-Agentフィールドは、リクエストを発行するユーザー・エージェントの情報を含んでいます。このフィールドは必ず要求されているわけではありませんが、ユーザー・エージェントはリクエスト中にこれを含めるべきです。Serverフィールドのときと同様の書式を用います。通常、アプリケーションを特定するのに重要な順番で、プロダクトのトークンがリストされます。
	書式	User-Agent= "User-Agent" ":" 1＊ (product ｜ comment)
	例	User-Agent：Emacs-W3/2.2.16 URL/1.359 (X11；linux)
WWW-Authenticate		Response-HeaderのWWW-Authenticateフィールドは、401 (Unauthorized) のレスポンス・メッセージに含まれていなくてはなりません。このフィールドは、Request-URIに対応している認証のスキームやパラメータを示す、少なくとも一つの問いかけによって、構成されていなくてはなりません。
	書式	WWW-Authenticate= "WWW-Authenticate" ":" 1＃challenge

表10-18②　HTTP/1.0のヘッダ・フィールドの書式一覧

Entity-Headerフィールドは、Entity-Bodyに関するオプショナルなメタ情報を定義します。もしボディがなければ、リクエストで指定されるリソースについて定義します。

```
Entity-Header = Allow                    ; (表10-18参照)
              | Content-Encoding         ; (表10-18参照)
              | Content-Length           ; (表10-18参照)
              | Content-Type             ; (表10-18参照)
              | Expires                  ; (表10-18参照)
              | Last-Modified            ; (表10-18参照)
              | extension-header         ; 拡張ヘッダ
Extension-header=HTTP-header
```

Extension-headerにより、プロトコルの変更なしにEntity-Headerフィールドの追加が可能ですが、それらは受信側が認識することを想定はしていません。認識できないヘッダ・フィールドは受信側で無視されるか、プロキシーによって転送されることになっています。

(2) Entity-Body

HTTPのリクエストやレスポンスで送られるエンティティ・ボディは、Entity-Headerフィールドで定義されたフォーマットとエンコーディングにしたがいます。エンティティ・ボディは、リクエストとレスポンスのどちらでも使用されることがあります。

```
Entity-Body=*OCTET
```

リクエストのメソッドがエンティティ・ボディを要求するときだけ、エンティティ・ボディはリクエスト・メッセージに含まれます。リクエストのエンティティ・ボディの有無は、リクエストのメッセージ・ヘッダの中のContent-Lengthフィールドが入っていることでわかります。エンティティ・ボディを含むHTTP/1.0リクエストは、有効なContent-Lengthフィールドをもっていなくてはなりません。

一方、レスポンス・メッセージでは、エンティティ・ボディがメッセージに含まれるかどうかは、リクエストのメソッドとステータス・コードで決まります。HEADのリクエスト・メソッドに対するすべてのレスポンスは、ボディを含んではいけません。また、すべての1xx（情報）、204（No Content）、304（Not Modified）のレスポンスは、ボディを含んではいけないか、Content-Lengthフィールドの値がゼロでなくてはなりません。

① エンティティ・ボディのタイプ

Entity-Bodyがメッセージに含まれるときは、そのボディのデータ・タイプはContent-TypeとContent-Encodingのフィールドを通して決まります。それら

は2段階のエンコーディングを行うことを示しています。

```
Entity-Body：= Content-Encoding (Content-Type (data))
```

Content-Typeは、下層のデータのメディア・タイプを定めます。Content-Encodingは、要求されたリソースの属性であり、通常、データ圧縮に用いるタイプに適用される、追加のコンテント・コーディングを示します。Content-Encodingのデフォルトは「なし」です。

エンティティ・ボディを含むどのようなHTTP/1.0メッセージも、そのボディのメディア・タイプを示すContent-Typeヘッダ・フィールドを含まなくてはなりません。もしそれが与えられなければ、Simple-Responseのメッセージのときのように、そのコンテンツのチェックを行うか、そのURLの拡張子から、メディア・タイプを推測します。それでも受信側で判別できないときは、"application/octet-stream" のタイプで処理することになります。

② エンティティ・ボディの長さ

エンティティ・ボディがメッセージに含まれるとき、そのボディの長さは次の二つの方法のどちらか一つで決められます。もしContent-Lengthのヘッダ・フィールドがあれば、バイト数で示されるその値は、エンティティ・ボディの長さを表します。そうでない場合は、サーバのコネクションの切断をもって、エンティティ・ボディの長さとします。

リクエストの場合、コネクションの切断を用いてそのボディの終了を示すことはできません。なぜなら、それではサーバがレスポンスを返すことが不可能になってしまうからです。エンティティ・ボディを含むHTTP/1.0のリクエストは、有効なContent-Lengthフィールドを含んでいなければなりません。もしリクエストにエンティティ・ボディが含まれておらず、Content-Lengthもなく、サーバが他のフィールドからその長さを計算することもできない場合には、サーバはレスポンスとして400 (Bad Request) を返すことになります。

⑨ メソッド (Method)

HTTP/1.0では、共通のメソッドとして、GET、HEAD、POSTの三つが定義されています。このセットは拡張されることがありますが、これらと同じ内容を含むことはありません。

(1) GETメソッド

GETメソッドは、Request-URIで示されるいかなる情報もエンティティのかたちで取得するということを意味します。もしRequest-URIがデータを作成するプログラムをさしているなら、レスポンスのエンティティとして返されるのは、そのプログラムが出力するデータであって、たまたまそれがそのプログラムの出力と同一でないかぎり、プログラム自身のソース・リストではありません。

もしリクエスト・メッセージにIf-Modified-Sinceのフィールドがあれば、GET

メソッドの意味がconditional GET（条件つきGET）に変わります。conditional GETメソッドは、If-Modified-Sinceヘッダで与えられる日付以降にリソースが変更されたときだけ、指定されたリソースをリクエストするというものです。この理由は、キャシュされたエンティティを用いることで、重複したリクエストや、不必要なデータ転送を防ぎ、ネットワーク・リソースの使用を減らすことにあります。GETメソッドの例を図10-32に示します。

(2) HEADメソッド

HEADメソッドは、サーバがレスポンス時にEntity-Bodyを何も返さないこと以外はGETメソッドと同じです。HEADリクエストに対するレスポンスのHTTPヘッダに含まれるメタ情報は、GETメソッドに対して送られる情報と同一でなくてはなりません。

このHEADメソッドは、Entity-Bodyを送らずに、Request-URIで指定されるリソースに関するメタ情報を取得するために用いられます。この方法は、ハイパーテキストのリンクの有効性や、接続性、変更情報などを知るために、用いられます。

HEADでは、GETのときのようにconditional HEADというものはありません。もしIf-Modified-Sinceのヘッダ・フィールドがHEADリクエストに含まれていても、無視されます。

HEADメソッドの例を図10-33に示します。

```
GET / HTTP/1.0                                                    ──→ Request-Line

HTTP/1.0 200 OK                                                   ──→ Status-Line
Date : Wed,03 Jul 1996 08 : 35 : 00 GMT
Server : Apache/1.1.1
Content-type : text/html                                          } HTTPヘッダ
Content-length : 435
Last-modified : Thu,18 Jan 1996 23 : 42 : 04 GMT
                                                                  ──→ CRLF
<HTML><HEAD><TITLE>
Internet 1996 World Exposition
</TITLE>
<meta http-equiv= "refresh" content= "20 ; URL=/bandwidth.cgi" >
</HEAD>

<BODY BGCOLOR= "#000000" TEXT= "#ffffff" LINK= "#d98719" VLINK= "#97694F" >
<center>
<a href= "bandwidth.cgi">
<img alt= "[Click Here For Our Low-Bandwidth Portal]"                                  } Entity
        src= "Images/train large3.jpg" width= "465" height= "349" border= "0" hspac    -Body
vspace= "30" align= "center" ></a>
</center><br clear>

</body>
</HTML>
```

図10-32　GETメソッドの例

(3) POSTメソッド

POSTメソッドは、リクエストに含まれるエンティティを、Request-Line中のRequest-URIで示されるリソースの付属として受理するよう、目的のサーバに要求するときに用いられます。

POSTメソッドは次のような機能を統一的な方法で処理します。

① 存在するリソースの注釈

② メッセージをニュース・グループ、メーリング・リストにPOST（投稿、送信）する

③ データを処理するプロセスに対して、フォームなどのデータのかたまりを送る

④ 追加操作によるデータベースの拡張

POSTメソッドによる実際の機能は、サーバによって決まるかまたはふつうはrequest-URIによって決まります。ファイルがそれを含んでいるディレクトリに付属したり、ニュース記事がPOSTされるニュース・グループに、レコードがデータベースにそれぞれ付属するように、POSTされたエンティティはそのURIに付属します。

POSTは成功しても、オリジン・サーバ上のリソースとして作成されたり将来の参照としてアクセス可能になるエンティティを、要求しません。POSTによって行われたアクションは、URIで示されるリソース中で終了しないかもしれません。この場合、結果が入っているエンティティを含むレスポンスの有無によって、200（OK）または204（No Content）のどちらかがステータス・コードになります。

もし、リソースがオリジン・サーバ上でつくられるなら、そのレスポンスは201（Created）となり、リクエストのステータスを記述し、新しいリソースを参照するエンティティを含んでいます。

すべてのHTTP/1.0のPOSTのリクエストにおいて、有効なContent-Lengthが必要です。もしHTTP/1.0のサーバがリクエスト・メッセージの中身の長さを特定することができない場合には、サーバは400（Bad Request）のメッセージを返さなくてはなりません。

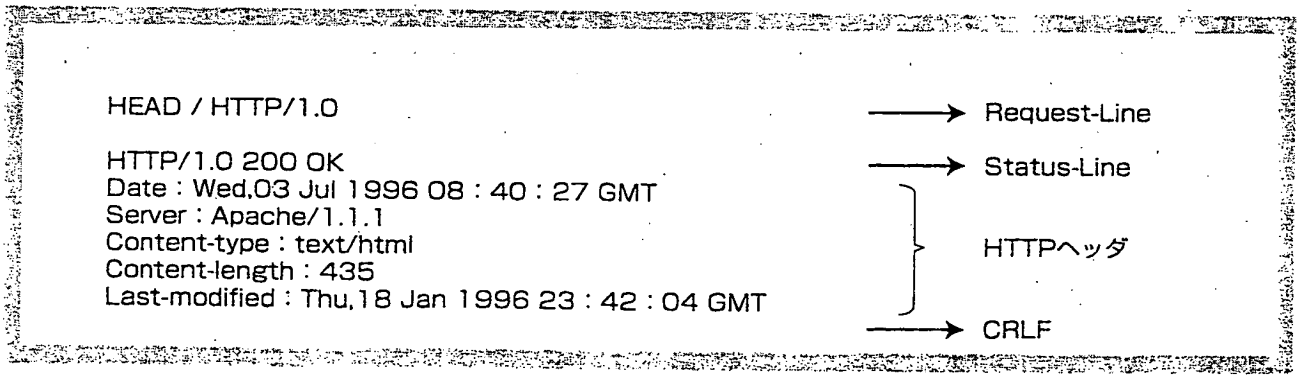


図10-33 HEADメソッドの例

なおアプリケーションは、POSTに対するレスポンスをキャッシュしてはいけません。

POSTメソッドの例を図10-34に示します。

⑩ ヘッダ・フィールドの定義

HTTP/1.0のヘッダ・フィールドで共通に使用されるすべての文法とその意味を表10-18に示します。

⑪ アクセス認証（Access Authentication）

(1) アクセス認証の概要

HTTPは、サーバがクライアントのリクエストに対して問いかけを行い、クライアントは認証情報を送るという、問いかけと返答による単純な認証メカニズムを用いています（図10-35）。ここでは、認証スキームを特定するための、拡張可能なトークン（ケース・インセンシティブ）と必要なパラメータが用いられます。

```
auth-scheme    = token
auth-param     = token "=" quoted-string
```

ユーザー・エージェントの認証の問いかけを行うため、オリジン・サーバは401（Unauthorized）のレスポンス・メッセージを用います。このレスポンスは、要求されたリソースに対応した少なくとも一つの問いかけを含む、WWW-Authenticateフィールドをもつ必要があります。

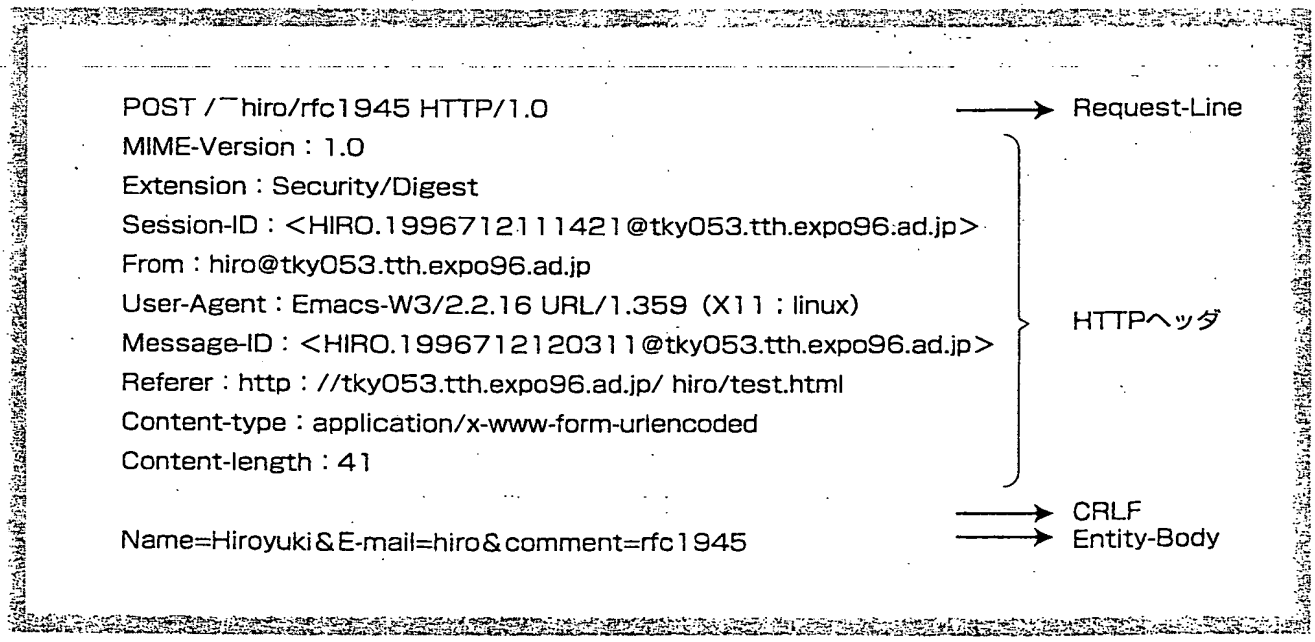


図10-34　POSTメソッドの例

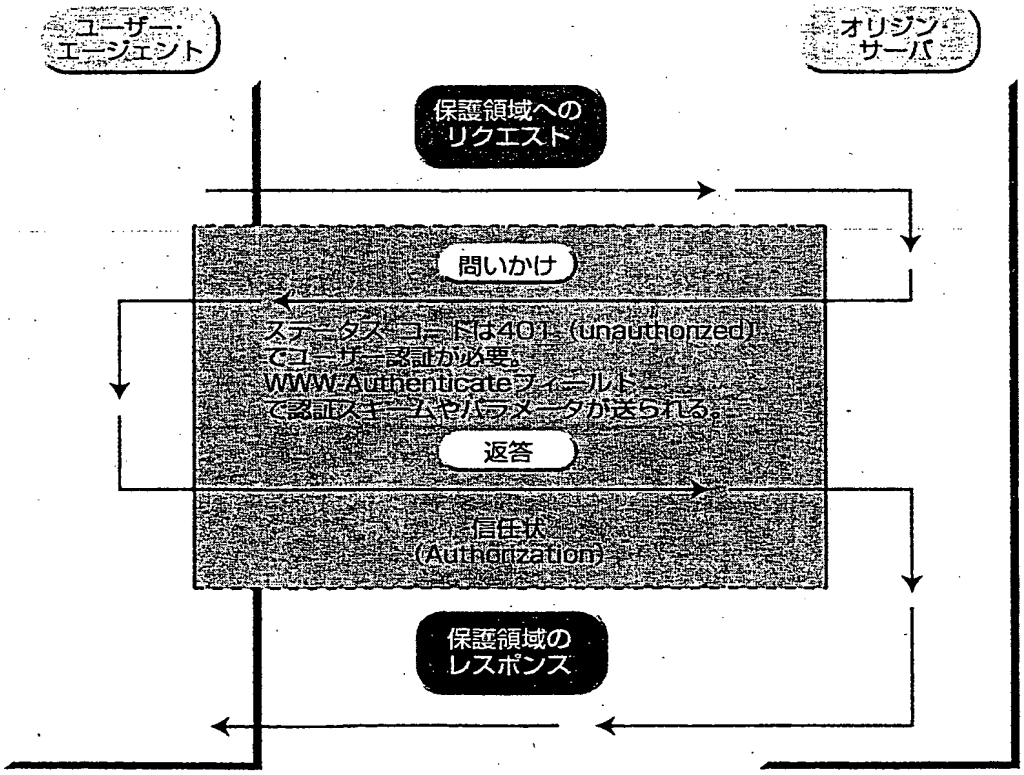
```
challenge       = auth-scheme 1*SP realm* ( "," auth-param)
realm           = "realm" "=" realm-value
realm-value     = quoted-string
```

問いかけを発行するすべての認証スキームは、このrealmアトリビュート（ケース・インセンシティブ）が必要です。アクセスされるサーバの通常のルートのURLとの組み合わせで、このrealm値（ケース・センシティブ）は保護領域を定めます。別々の認証スキームと認証データベースを用いた保護領域の分割を、サーバのリソースに対して行うこともできます。

ユーザー・エージェントは、401のレスポンスを受けとった後に通常、次のリクエストの中に認証のヘッダ・フィールドを含めることで、サーバに認証してもらいます。認証フィールドの値は、リソースの領域に対するユーザー・エージェントの認証情報を含んだ信任状で構成されています。

```
credentials = basic-credentials
            | (auth-scheme#auth-param)
```

ユーザー・エージェントによって信任状が自動的に適用することができる領域



網掛け の部分がHTTPの認証メカニズム部分

図10-35　アクセス認証のメカニズム

ポイント 図10-35

❶ 保護された領域へのアクセスに対して、オリジン・サーバがユーザー・エージェントへ認証の問いかけを行う

オリジン・サーバはステータス・コード401を返す

は、それぞれの保護領域によって決められます。もし、その前のリクエストが承認されているならば、認証スキームやパラメータなどで決められるある時間内は、同じ信任状が再利用されます。サーバは、リクエストの中の信任状を受諾できないときは、403（Forbidden）のレスポンスを返します。

プロキシーは、WWW-AuthenticateやAuthorizationのヘッダについては、何も手を加えずに、そのまま転送しなくてはなりません。また、承認を含むリクエストのレスポンスをキャッシュされないし、してもいけません。それらがキャッシュされては、セキュリティの点で認証の意味がなくなってしまうからです。

(2) 基本認証スキーム

基本認証スキームでは、ユーザー・エージェントはそれぞれの領域に対してuser-IDとpasswordで認証を受けます。リクエストURIの保護領域に対するuser-IDとpasswordが有効であったときにかぎり、サーバはそのリクエストを許可します。オプションの認証パラメータはありません。保護領域内のURIに対する認証できないリクエストを受けたら、サーバは次のような問いかけをユーザー・エージェントに返します。

```
WWW-Authenticate：Basic realm= "SECRET_PAGES"
```

ここで、"SECRET_PAGES" とは、そのRequest-URIの保護された領域を示すためにサーバによって決められた文字列です。

☞BASE64に関しては、4.3.4のMIMEの項を参照してください。

ユーザー・エージェントは認証を受けるため、信任状の中で、BASE64でエンコードされた文字列のuser-IDとpasswordを、シングル・コロン（":"）で区切って送ります。

```
basic-credentials  = "Basic" SP basic-cookie
basic-cookie       = 〈BASE64 エンコードされたuser-IDとuserid-password〉
userid-password    = [token] ":" *TEXT
```

もしユーザー・エージェントがuser-ID "Hirake" とパスワード "Goma Goma" を使いたければ、次のようなヘッダ・フィールドを使用することになります。

```
Authorization：Basic SG1yYWt1IEdvbWEgR29tYQ==
```

追加機能

すべての環境に実装されているわけではありませんが、追加のヘッダ・フィールドの名前を次に示します。

Accept、Accept-Charset、Accept-Encoding、Accept-Language、
Content-Language、Link、MIME-Version、Retry-After、Title、URI

HTTP/1.1

HTTP/1.0の次のバージョンであるHTTP/1.1（RFC2068）❶とそれ以前のバージョンのいちばんの違いは、デフォルトのHTTP接続の形態として、同じサーバに対して一つのコネクションで複数のデータを取得するパーシステント・コネクション（Persistent connection）が導入されたことです。

❶…1997年1月3日にRFC化された。

HTTP/1.0以前では、一つのリクエストごとにTCPのコネクションをオープンし、データ転送し、コネクションをクローズしていました。しかし、インライン・イメージや他の関連するデータは、きわめて短い時間に同じサーバに対していくつものリクエストをしばしば発行し、HTTPサーバやネットワークの負荷を増やしてしまいます。パーシステント・コネクションはこれを改善します。

HTTPのパーシステント・コネクションの利点としては次のようなことが考えられます。

① TCPコネクションのオープンとクローズを減らすので、CPU時間を節約し、TCPのコントロール・ブロックに使用されるメモリも節約されます。

② HTTPのリクエストとレスポンスは、一つのコネクションでパイプライン処理することが可能になります。パイプラインを使用すれば、クライアントは個々のレスポンスを待つことなしに、複数のリクエストを送ることができ、待ち時間なしで、一つのTCPコネクションを有効に使用することができます。

③ TCPのオープンにともなうパケットの数を減らし、ネットワークの混雑を減らすことができます。

このほかにHTTP/1.1では、OPTIONS、PUT、DELETE、TRACEという新しいメソッドが追加されます。

10.5 ATMフォーラムのマルチメディア情報通信プロトコル

ATMネットワークとマルチメディア通信

ATMネットワークは、次のような特徴をもっています。

① 交換機（ATMスイッチ）を使用する交換型のネットワークである

② ハードウェアによる処理を中心とするため高速広帯域である

③ ディジタル化されたあらゆる種類のデータを搬送できる

④ アプリケーションやサービスが獲得した通信路を占有できる

⑤ アプリケーションの必要性に応じたサービス品質を指定できる

⑥ 物理的なネットワーク形態に依存しない、論理的なネットワークを構成できる

本書の内容に関するご質問は、小社オープン ネットワーク編集部まで、封書（返信用切手同封のこと）にてお願い致します。
電話によるお問い合わせには、応じられません。
なお、本書の範囲を越える質問に関しては、お答えできない場合もあります。
●落丁・乱丁本は、送料当社負担にてお取り替え致します。
お手数ですが、小社営業部までご返送ください。

ポイント図解式
通信プロトコル事典

1996年11月11日　初版発行
1997年4月25日　第一版第3刷発行

監修者　笠野 英松（かさの ひでまつ）
編　者　マルチメディア通信研究会
発行人　橋本 孝久
編集人　三橋 昭和
発行所　株式会社アスキー
〒151-24　東京都渋谷区代々木4-33-10
振　替　00140-7-161144
大代表　(03)5351-8111
出版営業部　(03)5351-8194（ダイヤルイン）
オープン ネットワーク編集部　(03)5351-8121（ダイヤルイン）

制　作　有限会社 シンクス
印　刷　株式会社 加藤文明社

カバーデザイン　鏡川筆山

Printed in Japan

ISBN4-7561-0269-7
定価 5,631円＋税＝5,913円

●1190392